**Use Multiline JSX in a Component**

In this lesson, you will learn some common ways that JSX and React components work together. You'll get more comfortable with both JSX and components, while picking up some new tricks.

Take a look at this HTML:

<blockquote> <p> The world is full of objects, more or less interesting; I do not wish to add any more. </p> <cite> <a target="\_blank" href="https://en.wikipedia.org/wiki/Douglas\_Huebler"> Douglas Huebler </a> </cite> </blockquote>

How might you make a React component that renders this HTML?

Select **QuoteMaker.js** to see one way of doing it.

The key thing to notice in QuoteMaker is the parentheses in the return statement, on lines 6 and 18. Until now, your render function returnstatements have looked like this, without any parentheses:

return <h1>Hello world</h1>;

However, a multi-line JSX expression should always be wrapped in parentheses! That is why QuoteMaker's return statement has parentheses around it.

# Use a Variable Attribute in a Component

Take a look at this JavaScript object named redPanda:

const redPanda = { src: 'https://upload.wikimedia.org/wikipedia/commons/b/b2/Endangered\_Red\_Panda.jpg', alt: 'Red Panda', width: '200px };

How could you render a React component, and get a picture with redPanda's properties?

Select **RedPanda.js** to see one way to do it.

Note all of the curly-brace JavaScript injections inside of the render function! Lines 16, 17, and 18 all use JavaScript injections.

You can, and often will, inject JavaScript into JSX inside of a render function.

# Put Logic in a Render Function

A render() function must have a returnstatement. However, that isn't all that it can have.

A render() function can also be a fine place to put simple calculations that need to happen right before a component renders. Here's an example of some calculations inside of a renderfunction:

class Random extends React.Component { render() { // First, some logic that must happen // before rendering: const n = Math.floor(Math.random() \* 10 + 1); // Next, a return statement // using that logic: return <h1>The number is {n}!</h1>; } }

Watch out for this common mistake:

class Random extends React.Component { // This should be in the render function: const n = Math.floor(Math.random() \* 10 + 1); render() { return <h1>The number is {n}!</h1>; } };

In the above example, the line with the const ndeclaration will cause a syntax error, as is it should not be part of the class declaration itself, but should occur in a method like render().

# Use a Conditional in a Render Function

How might you use a conditional statement inside of a render() function?

Select **TodaysPlan.js** to see one way of doing it.

Notice that the if statement is located insideof the render function, but before the returnstatement. This is pretty much the only way that you will ever see an if statement used in a render function.

# Use this in a Component

The word this gets used in React a lot!

You are especially likely to see this inside of the body of a component class declaration. Here's an example:

class IceCreamGuy extends React.Component { get food() { return 'ice cream'; } render() { return <h1>I like {this.food}.</h1>; } }

In the code, what does this mean?

Once you have a guess, scroll down to see the answer.

...

...

...

...

...

...

The simple answer is that this refers to an instance of IceCreamGuy. The less simple answer is that this refers to the object on which this's enclosing method, in this case .render(), is called. It is almost inevitable that this object will be an instance of IceCreamGuy, but technically it could be something else.

Let's assume that this refers to an instance of your component class, as will be the case in all examples in this course. IceCreamGuy has two methods: .food and .render(). Since this will evaluate to an instance of IceCreamGuy, this.food will evaluate to a call of IceCreamGuy's .food method. This method will, in turn, evaluate to the string "ice cream."

Why don't you need parentheses after this.food? Shouldn't it be this.food()?

You don't need those parentheses because .food is a getter method. You can tell this from the get in the above class declaration body.

There's nothing React-specific about getter methods, nor about this behaving in this way! However, in React you will see this used in this way almost constantly.

this in JavaScript can be a difficult concept! Here is a good resource for [understanding thisin JavaScript](https://dmitripavlutin.com/gentle-explanation-of-this-in-javascript/).

# Use an Event Listener in a Component

Render functions often contain event listeners. Here's an example of an event listener in a render function:

render() { return ( <div onHover={myFunc}> </div> ); }

Recall that an event handler is a function that gets called in response to an event. In the above example, the event handler is myFunc().

In React, you define event handlers as methodson a component class. Like this:

class MyClass extends React.Component { myFunc() { alert('Stop it. Stop hovering.'); } render() { return ( <div onHover={this.myFunc}> </div> ); } }

Notice that the component class has two methods: .myFunc() and .render(). .myFunc()is being used as an event handler. .myFunc()will be called any time that a user hovers over the rendered <div></div>.

# Components Recap

Congratulations! You have finished the unit on React components.

React components are complicated. Their syntax is complicated, and the reasoning behind their syntax is especially complicated.

You have learned a lot about both their syntax and their reasoning. You have learned about component classes and component instances. You have learned about React.Component, and about the instructions that you must provide to a component class. You have learned how to import, and how to render a component instance.

You have been introduced to some common ways of using JSX in React components. You have rendered components using multiline JSX expressions, logic inside of the render function, a conditional statement, this, and an event listener.

You have spent a lot of time studying React components in isolation! Now, it's time to start learning how components fit into with the world around them.